# Supplementary Material for
# Representation Compensation Networks for Continual Semantic Segmentation

Chang-Bin Zhang [1] Jia-Wen Xiao [1] Xialei Liu [1] Ying-Cong Chen [2] Ming-Ming Cheng [1]
[1] TMCC, CS, Nankai University [2] HKUST

## 1. Overview

In Sec. 2, We firstly conduct experiments on classification [4,7] in continual learning. We describe the details of our method in Sec. 3, where we provide the pseudo-code for our proposed Pooled Cube Distillation. Then we discuss some of the characters in our method in Sec. 4, including the robustness of our method against different class orders, the impact of hyper-parameters ,and the ablation study about pooled knowledge distillation. More importantly, we explore our proposed representation compensation mechanism in Sec. 5. Lastly, we display more qualitative results in Sec. 6.

## 2. Continual Learning on Classification.

Our proposed representation compensation module can be easily integrated with many existing continual learning methods [1, 4, 7]. We conducted experiments integrating our representation compensation mechanism and two existing methods, EEIL [7] and PODNet [4]. We follow PODNet [4] and report Top-1 accuracy on ImageNet-Subset (100 classes in total) with 50 classes as the first task and the rest are equally divided into five tasks (step 1 - 5). As shown in Tab. 1, For both EEIL and PODNet baselines, our method improves over them by about 2% in average accuracy, respectively.

| Method | step 1 | step 2 | step 3 | step 4 | step 5 | Avg Gain |
|---|---|---|---|---|---|---|
| EEIL [7] | 74.27 | 70.03 | 68.45 | 64.62 | 61.62 | |
| + RC | **77.23** | **72.06** | **70.10** | **66.89** | **63.82** | 2.22 ↑ |
| PODNet [4] | 81.20 | 72.74 | 66.15 | 61.47 | 57.44 | |
| + RC | **81.90** | **74.77** | **70.05** | **64.22** | **60.04** | 2.40 ↑ |

Table 1. Experiments in Continual Classification. All experiments are conducted on the ImageNet-100.

## 3. Reproducibility

In this section, we describe more details about the loss functions. Then we provide the pseudo-code for our proposed pooled cube distillation.

**Algorithm 1** Pseudo-code of Pooled Cube Distillation in a PyTorch-like style.

```
# f_old: list of features from different stages of the
      old model
# f_new: list of features from different stages of the
      new model
# gamma: the hyper-parameters for the loss function
# loss_spatial: Pooled Cube distillation loss on
     spatial dimension
# loss_channel: Pooled Cube distillation loss on
     channel dimension

def pooledCube_KD(f_old, f_new):
    # define the average pooling size
    kernel_spatial = [4,8,12,16,20,24]
    kernel_channel = [3]

    # do PCD for different pairs of features
    for i, (x_old, x_new) in zip(f_old, f_new):

        #x_old: NxCxHxW
        PCD_old = hadamard_product(x_old, x_old)
        #x_new: NxCxHxW
        PCD_new = hadamard_product(x_new, x_new)

        loss_spatial = 0

        # multi-scale pooled cube distillation on
             spatial dimension
        for kernel in kernel_spatial:
            PCD_old = AvgPool2d(PCD_old, kernel)
            PCD_new = AvgPool2d(PCD_new, kernel)
            PCD_gap = (PCD_old - PCD_new).view(N,-1)
            PCD_gap = hadamard_product(PCD_gap, PCD_gap)

            loss_spatial += sqrt(PCD_gap.sum())

        loss_spatial /= len(kernel_spatial)

        # pooled cube distillation on channel dimension
        PCD_old = x_old.permute(0,2,1,3)
        PCD_new = x_new.permute(0,2,1,3)

        loss_channel = 0

        for kernel in kernel_channel:
            PCD_old = AvgPool2d(PCD_old, (kernel, 1))
            PCD_new = AvgPool2d(PCD_new, (kernel, 1))
            PCD_gap = (PCD_old - PCD_new).view(N,-1)
            PCD_gap = hadamard_product(PCD_gap, PCD_gap)

            loss_channel += sqrt(PCD_gap.sum())

        loss_channel /= len(kernel_channel)

    # compute the total loss
    loss = (loss_channel + loss_spatial) * gamma

    return loss
```

**Objective.** In the scenario of continual class semantic segmentation, to save the labeling cost, only the new classes

are labeled in the newly added training data, and the old classes are treated as the *background* class. Thus, this brings a great challenge in continual class semantic segmentation, semantic shift [2]. To solve this issue, we also apply the loss functions $L_{unce}$ and $L_{unkd}$ proposed by [2] as [2, 6] in our pipeline as the baseline. We refer to [2] for more details.

Specifically, let $C_t$ denotes the classes learned in step $t$. Thus, for the example $(x, y)$, the objective for learning new classes can be written as

$$L_{unce} = -\frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}}\log \hat{p}_t(i, y_i), \quad (1)$$

where $y_i \in \{0, C_t\}$ denotes the ground-truth in the label for the $i$-th pixel. And $\hat{p}_t(i)$ is modified from the predictions of current model $p_t(i)$, considering all old classes are *background*. The predicted scores for the old classes are summed to the *backgroud* class. The model is also supposed to maintain discrimination for old classes. Thus, the knowledge distillation objective can be denoted as

$$L_{unkd} = -\frac{1}{|\mathcal{I}|}\sum_{k\in\mathcal{C}}\sum_{i\in\mathcal{I}} p_{t-1}(i, k)\log \hat{p}_t(i, k), \quad (2)$$

where $p_{t-1}$ is the prediction of the old model, and $\mathcal{C}$ denotes all old classes and the *background* class. The $\hat{p}_t(i)$ is modified by predicted scores $p_t(i)$ of the current model. In this objective, all new classes are treated as the *background* class, and their predicted scores are summed to the *background* class.

In this work, the overall objective can be denoted as:

$$L = L_{unce} + \lambda L_{unkd}\cdot\sqrt{\frac{||C||}{||C_t||}} + \gamma(L_{skd} + L_{ckd}), \quad (3)$$

where $L_{skd}$ and $L_{ckd}$ denote the distillation loss function on spatial and channel dimensions, respectively. The $\lambda, \gamma$ are hyper-parameters to balance the different objectives. And the $||C||$ and $||C_t||$ denote the number of classes of all and current, respectively. In our experiments, we set the $\lambda$ as 100, and the $\gamma$ as 0.01. We discuss the impact of hyper-parameters in Sec. 4.2.

**Pooled Cube Distillation.** To further alleviate catastrophic forgetting, we design pooled cube distillation strategy on both spatial and channel dimensions. We display the pseudo-code in Alg. 1.

# 4. Discussion

## 4.1. Robustness to Class Order

To verify the impact of different class orders, we run different methods on five different orders on the 15-1 overlapped setting, which includes the ascending order and four random orders. The four random orders are provided by the code of PLOP [3]. Experimental results are shown in Tab. 2. We can observe that ILT [5], MiB [2], and SDR [6] are less stable to different orders with large variance. PLOP [3] improves over these methods by using multi-scale feature distillations. Thanks to the proposed mechanisms, Ours is much more robust to different orders and also obtains the best performance in terms of mIoU. The five orders are defined as:

$$\begin{aligned}
&A: \{[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15], [16], [17], [18], [19], [20]\},\\
&B: \{[0, 12, 9, 20, 7, 15, 8, 14, 16, 5, 19, 4, 1, 13, 2, 11], [17], [3], [6], [18], [10]\},\\
&C: \{[0, 13, 19, 15, 17, 9, 8, 5, 20, 4, 3, 10, 11, 18, 16, 7], [12], [14], [6], [1], [2]\}\\
&D: \{[0, 15, 3, 2, 12, 14, 18, 20, 16, 11, 1, 19, 8, 10, 7, 17], [6], [5], [13], [9], [4]\},\\
&E: \{[0, 7, 5, 3, 9, 13, 12, 14, 19, 10, 2, 1, 4, 16, 8, 17], [15], [18], [6], [11], [20]\}.
\end{aligned} \quad (4)$$

## 4.2. Impact of Hyper-parameters

As described in Sec. 3, there are two hyper-parameters $\lambda$ and $\gamma$ in our objective. We study the impact of these hyper-parameters in Tab. 3. Our method achieves the best performances when $\gamma = 0.005$ and $\gamma = 0.01$, with the selected $\gamma$, it can perform well within a relatively large range of $\lambda$ from 20 to 200. In our experiments, considering $lambda$ is set as 100 in [2], thus we apply the same hyper-parameters as [2]. We set $\lambda$ as 100 and $\gamma$ as 0.01.

## 4.3. Ablation study about knowledge distillation

**Impact of different pooling kernel sizes.** In the scenario of continual semantic segmentation, the pooling operation plays a key role in the distillation mechanism. In our distillation mechanism, we use the multi-scale average pooling with kernel size in $\mathcal{M} = \{4, 8, 12, 16, 20, 24\}$. We study the impact of different pooling kernel sizes in Tab. 4. Experimental results demonstrate that if only one window size is used when the pooling window size is too small or too larger, the performance will be worse. We analyze that if the pooling window size is relatively small, when aggregating information for the current pixel, sufficient information of neighbors is not able to be considered, so the negative impact of noise cannot be effectively suppressed. When the pooling window size is relatively large, aggregating information for the current pixel can bring unrelated noise to the current pixel, therefore the performance is worse as well. When we combine multi-scale window sizes, the mIoU is stable at very high performance, therefore we use all scales as shown in Tab. 4 in stead of choosing the optimal scales.

**Distillation on different layers.** We explore the impact of our proposed pooled cube distillation on different intermediate layers. Experimental results are shown in Tab. 5, which demonstrates that distillation on all layers outperforms the baseline without distillation by 21.7% in terms of mIoU. It

| Method | Task | overlapped | disjoint |
|---|---|---|---|
| ILT [5] | A | 9.20 | 7.90 |
| | B | 16.74 | 20.65 |
| | C | 12.16 | 6.37 |
| | D | 11.49 | 10.85 |
| | E | 15.60 | 13.77 |
| | | 13.04 ± 2.76 | 11.91±5.05 |
| MiB [2] | A | 32.20 | 39.9 |
| | B | 20.15 | 23.68 |
| | C | 36.05 | 34.25 |
| | D | 38.91 | 40.55 |
| | E | 53.73 | 48.01 |
| | | 36.21±10.8 | 37.28±8.08 |
| SDR [6] | A | 44.39 | 45.68 |
| | B | 40.65 | 6.60 |
| | C | 46.36 | 34.31 |
| | D | 44.61 | 37.04 |
| | E | 41.72 | 45.15 |
| | | 43.55 ± 2.07 | 33.76 ± 14.29 |
| PLOP [3] | A | 54.60 | 46.50 |
| | B | 47.43 | 41.67 |
| | C | 53.43 | 48.00 |
| | D | 58.25 | 46.81 |
| | E | 47.20 | 37.86 |
| | | 52.18 ± 4.28 | 44.17 ± 3.82 |
| Ours | A | 59.40 | 54.70 |
| | B | 54.05 | 53.26 |
| | C | 55.63 | 49.53 |
| | D | 55.29 | 55.53 |
| | E | 63.19 | 56.07 |
| | | 57.51 ± 3.35 | 53.82 ± 2.34 |

Table 2. The mIoU(%) of the final step. We conduct experiments on different class orders on 15-1 overlapped. The purple denotes the mean mIoU(%) and standard variance over five different class orders.

| $\lambda$ \ $\gamma$ | 0.0001 | 0.001 | 0.005 | 0.01 | 0.05 | 0.1 |
|---|---|---|---|---|---|---|
| 1 | 35.4 | 39.8 | 46.3 | 49.3 | 46.5 | 42.8 |
| 10 | 44.3 | 49.3 | 52.1 | 51.0 | 46.5 | 44.7 |
| 20 | 49.0 | 56.9 | 57.6 | 56.1 | 50.0 | 47.8 |
| 50 | 48.5 | 57.4 | **59.7** | 59.1 | 53.6 | 50.6 |
| 100 | 42.9 | 55.0 | **59.4** | **59.4** | 55.5 | 50.8 |
| 150 | 52.6 | 52.6 | 58.2 | 58.9 | 55.4 | 50.7 |
| 200 | 50.0 | 50.0 | 57.8 | 58.3 | 55.1 | 51.0 |

Table 3. Impact of different hyper-parameters. All experiments are conducted on the 15-1 overlapped setting on PASCAL VOC 2012 dataset. We select the $\lambda$ as 100 and $\gamma$ as 0.01 in our experiments.

| 4 | 8 | 12 | 16 | 20 | 24 | mIoU(%) |
|---|---|---|---|---|---|---|
| ✓ | | | | | | 55.1 |
| | ✓ | | | | | **56.2** |
| | | ✓ | | | | **56.2** |
| | | | ✓ | | | 55.4 |
| | | | | ✓ | | 54.7 |
| | | | | | ✓ | 53.7 |
| ✓ | ✓ | | | | | 55.8 |
| ✓ | ✓ | ✓ | | | | 56.1 |
| ✓ | ✓ | ✓ | ✓ | | | **56.2** |
| ✓ | ✓ | ✓ | ✓ | ✓ | | 56.1 |
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | 56.1 |

Table 4. Impact of different average pooling kernel sizes in our proposed pooled cube distillation mechanism. All experiments are conducted on 15-1 overlapped on PASCAL VOC 2012 dataset using PLOP framework.

| *layer 1* | *layer 2* | *layer 3* | *layer 4* | *decoder* | 15-1 |
|---|---|---|---|---|---|
| | | | | | 36.1 |
| ✓ | | | | | 33.6 |
| | ✓ | | | | 34.0 |
| | | ✓ | | | 39.7 |
| | | | ✓ | | 47.2 |
| | | | | ✓ | 54.1 |
| ✓ | ✓ | | | | 32.8 |
| ✓ | ✓ | ✓ | | | 34.0 |
| ✓ | ✓ | ✓ | ✓ | | 46.6 |
| | | | ✓ | ✓ | 55.3 |
| | | ✓ | ✓ | ✓ | 56.6 |
| | ✓ | ✓ | ✓ | ✓ | 57.4 |
| ✓ | ✓ | ✓ | ✓ | ✓ | **57.8** |

Table 5. Ablation study about distillation mechanism at different stages. All experiments are conducted on the 15-1 overlapped setting on PASCAL VOC 2012 without RC module.

is interesting that distillation on the decoder gives the largest boost compared to other layers, which may be due to the high-level semantic information contained in the decoder. Thanks to deep supervision, the effect of gradient vanishing can be alleviated, and fusing distillation from all layers can further improve the performance. Therefore we use distillation on all layers in our work.

## 5. Exploring Representation Compensation

Previously, we claimed that the left branch has the function of remembering the old knowledge, playing a role of great significance in preventing catastrophic forgetting. In Fig. 2, we let the left branch account for $99.95\%, 75\%, 50\%, 25\%, 0.05\%$ during fusion of training process to observe the model's ability to remember old knowledge, *i.e.,* the performance over old classes. In order to ensure the fairness of the experiment and easy observation,

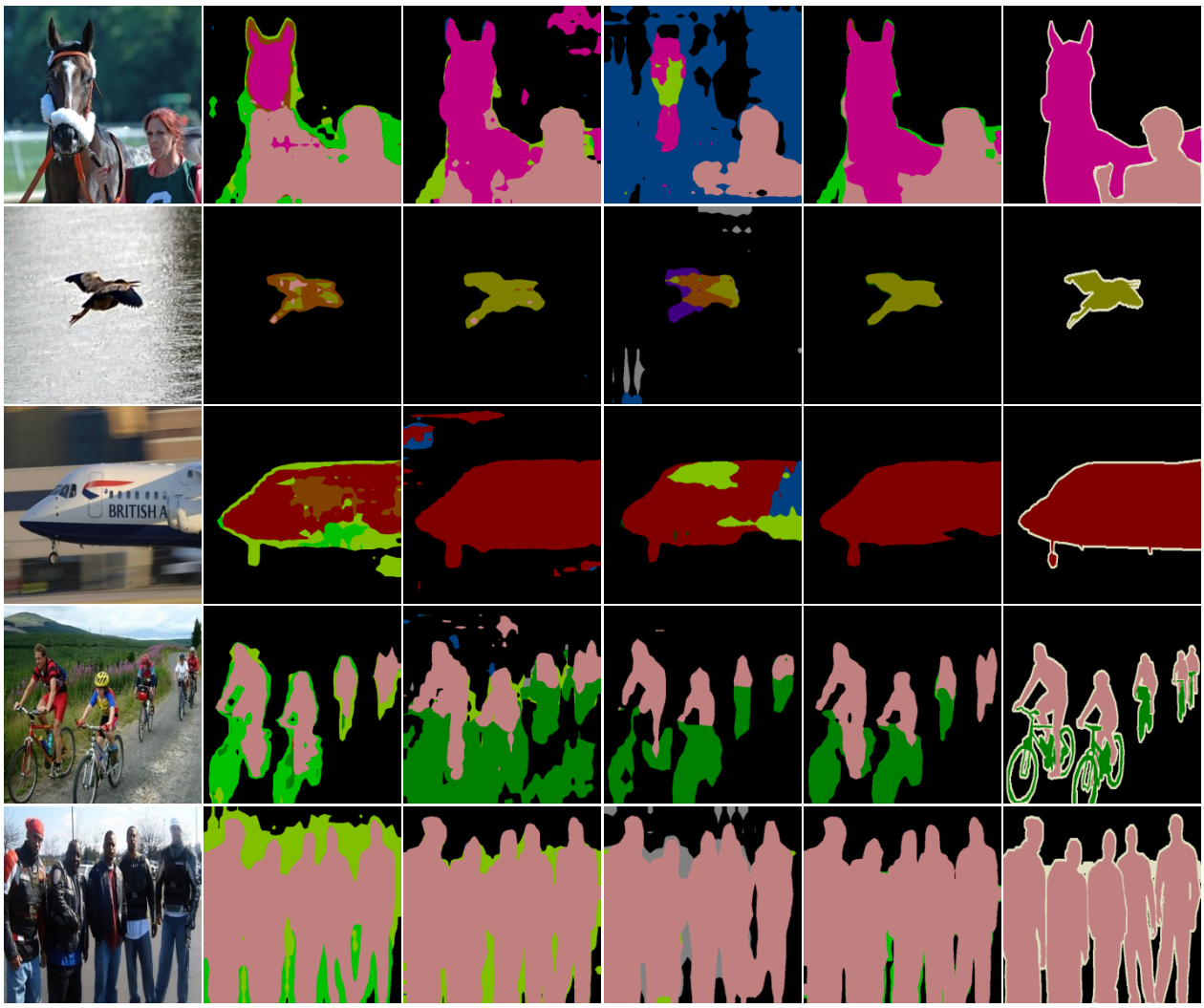


Figure 1. Visualization results for different methods.

we only added RC-module based on Fine-tuning. And we set the learning rate as 0.0001 for training from step 1 to step 5. As shown in Fig. 2, with the weight increasing, the model gradually enhances the memory of old knowledge, indicating that the frozen branch can preserve the old knowledge. Thus, our training process can benefit from this characteristic.

## 6. More Qualitative Results

We display some visualization results in Fig. 1.

## 7. Future Work

In our current RC-module, the feature aggregation of two branches is obtained by linear weighting. The weights indicate the importance of the two branches. In our method, we simply set the weights of two branches to 0.5. We believe that it can achieve better performance by designing the feature aggregation method carefully. For example, in future work, we could explore learnable weights for two branches.

## References

[1] Hongjoon Ahn, Jihwan Kwak, Subin Lim, Hyeonsu Bang, Hyojun Kim, and Taesup Moon. Ss-il: Separated softmax for incremental learning. In *Int. Conf. Comput. Vis.*, pages 844–853, 2021. 1

[2] Fabio Cermelli, Massimiliano Mancini, Samuel Rota Bulo, Elisa Ricci, and Barbara Caputo. Modeling the background for incremental learning in semantic segmentation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 9233–9242, 2020. 2, 3, 4

[3] Arthur Douillard, Yifu Chen, Arnaud Dapogny, and Matthieu Cord. Plop: Learning without forgetting for continual semantic segmentation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2021. 2, 3, 4

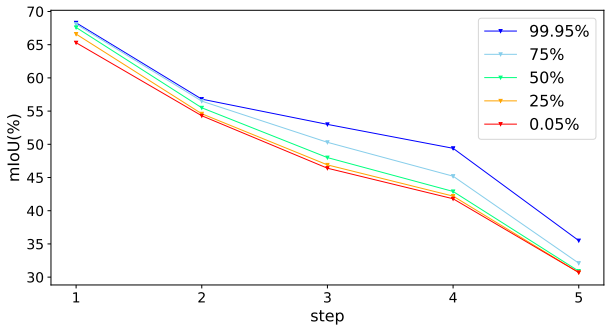


Figure 2. The mIoU(%) for old classes. We set different weighting parameters (99.95%, 75%, 50%, 25%, 0.05%) for the frozen branch during aggregating features from two branches. As the weight increases, the model presents a tendency to keep the memory of old knowledge. All experiments are conducted on 15-1 overlapped setting on PASCAL VOC 2012 dataset.


[4] Arthur Douillard, Matthieu Cord, Charles Ollion, Thomas Robert, and Eduardo Valle. Podnet: Pooled outputs distillation for small-tasks incremental learning. In *Eur. Conf. Comput. Vis.*, volume 12365, pages 86–102, 2020. 1

[5] Umberto Michieli and Pietro Zanuttigh. Incremental learning techniques for semantic segmentation. In *Int. Conf. Comput. Vis. Worksh.*, 2019. 2, 3

[6] Umberto Michieli and Pietro Zanuttigh. Continual semantic segmentation via repulsion-attraction of sparse and disentangled latent representations. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2021. 2, 3, 4

[7] Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. Large scale incremental learning. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019. 1